香港中文大學
The Chinese University of Hong Kong

# Open-Source, Python-Based Redevelopment of the ChemShell Multiscale QM/MM Environment


*[1] J. Chem. Theory Comput. 2019, 15, 1317−1328*

*[2] WIREs Comput Mol Sci 2014, 4:101– 110. doi: 10.1002/wcms.1163*

*[3] Phys. Chem. Chem. Phys. 2023, 25:21816–21835. doi: 10.1039/d3cp00648d*



Jingyi Liu


*04/06/2026*

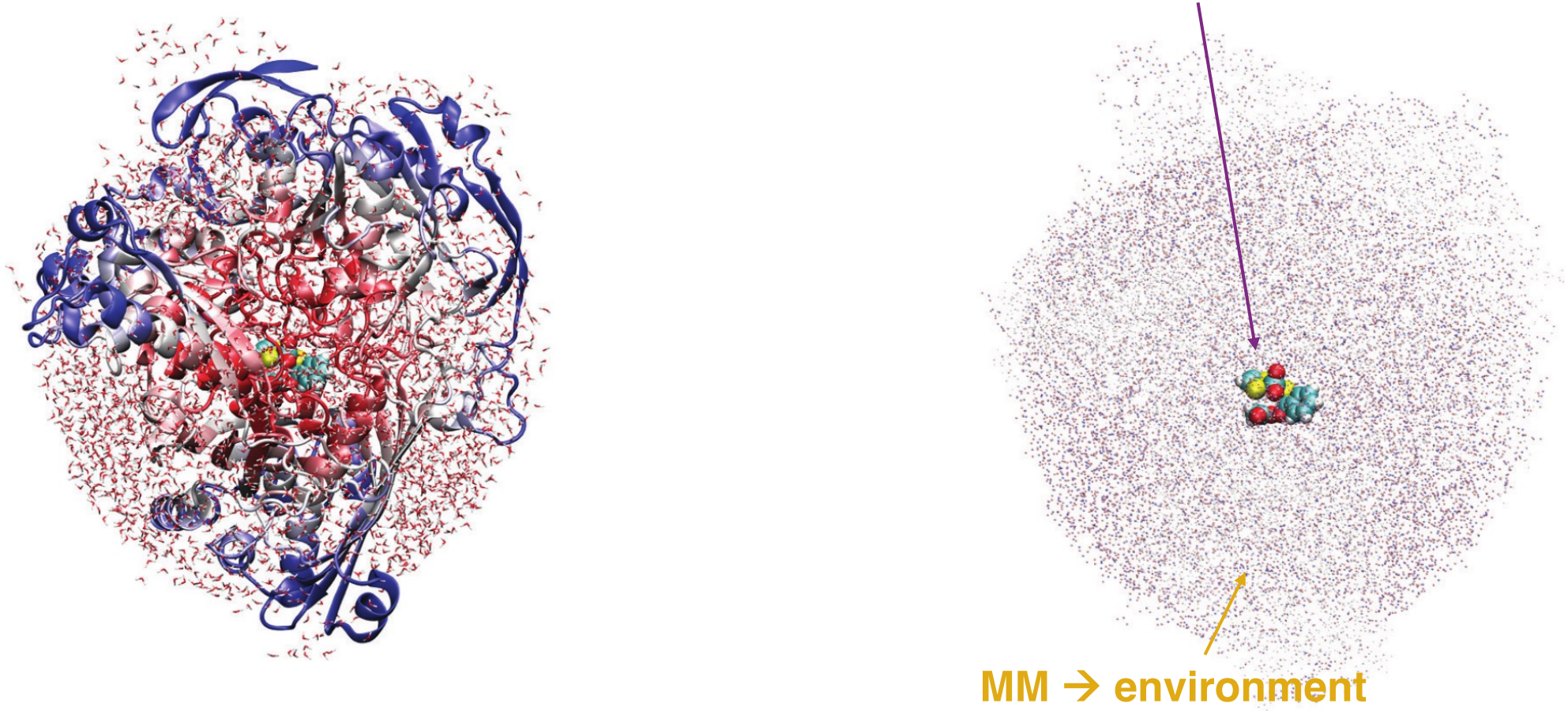
# Outline

- **Background**
- **The Original ChemShell package**
- **Python-based Redevelopment ChemShell (Py-ChemShell)**
- **Recent Progress**
- **Conclusion**

# Research Background

# Background

## QM / MM Treatment




QM/MM representation (left) of aldehyde oxidoreductase (right)[1]

[1] WIREs Comput Mol Sci 2014, 4:101– 110. doi: 10.1002/wcms.1163

# The Original Design of ChemShell

## QM / MM energy sc

- Subtractive
- Additive $E(\mathcal{S})$

## QM / MM embeddin

- Mechanical embeddir
- **Electrostatic embed**
- Polarized MM embed

**TABLE 1** | External QM and MM Codes that can be Interfaced to ChemShell

| | QM/MM features | | | HPC features | |
|---|---|---|---|---|---|
| | Electrostatic embedding | Shell model embedded cluster calculations | Multiple state interface | Directly linked interface | Task-farming parallelization (directly linked) |
| QM codes | | | | | |
| GAMESS-UK | X | X | | X | X |
| NWChem | X | X | | X | |
| FHI-AIMS[1] | X | X | | X | |
| DALTON | X | | | X | X |
| LS-DALTON[1] | X | | | X | X |
| DMol3 | X | | | | |
| TURBOMOLE | X | | | | |
| ORCA | X | | | | |
| GAMESS (US) | | | | | |
| Q-Chem | X | | | | |
| Gaussian[2] | X | | X | | |
| Molpro | X | | X | | |
| MNDO | X | | X | X | |
| MOPAC | X | | | | |
| MM codes | | | | | |
| DL_POLY 2[3] | X | | | X | X |
| GULP | X | X | | X | X |
| GROMOS | X[4] | | | | |
| CHARMM | X[4] | | | | |

[1] WIREs Comput Mol Sci 2014, 4:101– 110. doi: 10.1002/wcms.1163
[2] J. Chem. Theory Comput. 2022, 18, 4601−4614

# The Original Design of ChemShell

## ➢ Boundary Treatment

**Covalent**

❖ **Link atom approach**

- adding hydrogen atoms to the QM region while force field terms in the MM calculation are selectively deleted
  → no interactions are double counted
- the charges on the MM atoms closest to the QM region are shifted to neighboring atoms
  → avoid overpolarization of the QM system
- Extra dipoles are then added to compensate for the dipoles created by the charge shifting scheme.
- The forces on the link atoms are projected onto real atoms after each gradient evaluation.

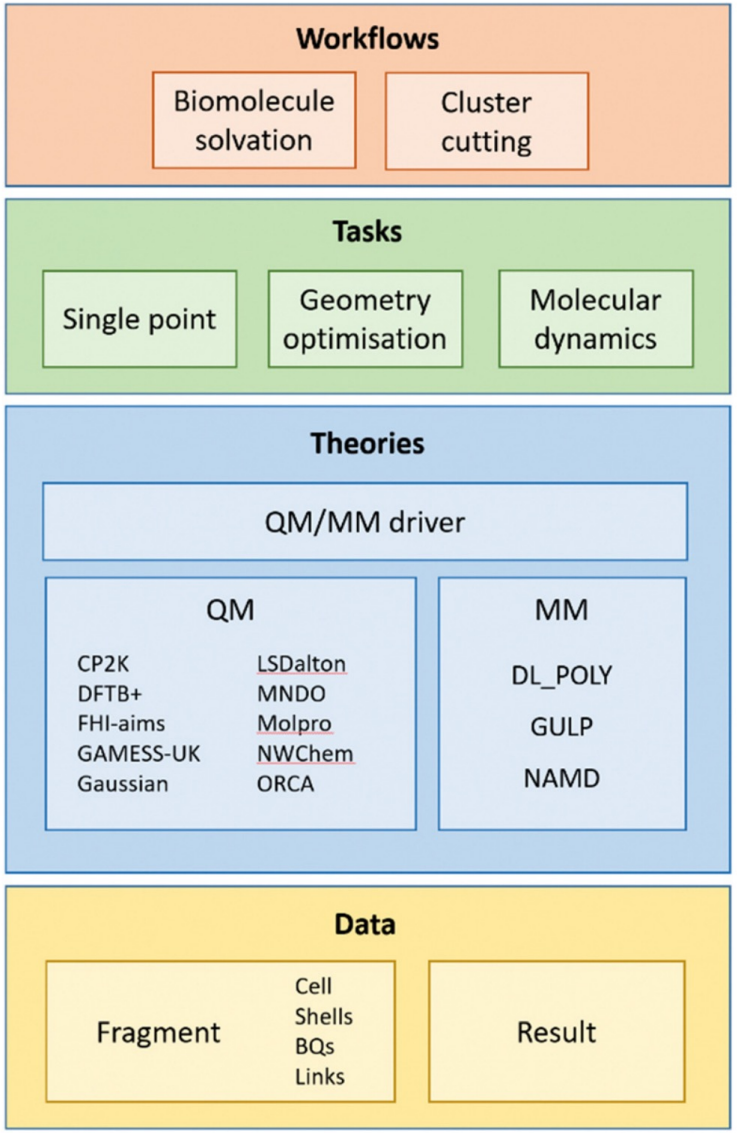
**Ionic**

❖ atomic, large-core pseudopotentials are placed on the MM atoms immediately surrounding the QM region
→ localize electrons in the QM region,
→ preventing QM atoms from spilling out into the MM



[1] WIREs Comput Mol Sci 2014, 4:101– 110. doi: 10.1002/wcms.1163

# The Original Design of ChemShell




Overview of ChemShell's modular framework

Handles the QM and MM boundary

Input: details of the system's geometry, periodicity, charges and other atomic attributes (PDB, xyz, etc.)



[3] Phys. Chem. Chem. Phys. 2023, 25:21816–21835. doi: 10.1039/d3cp00648d

# Python-based Redevelopment (Py-ChemShell)

The original package is written in Tcl language (Tcl-ChemShell).

- Python becomes one of the most popular programming language in computational chemistry.
- Python is more flexible than Tcl, with better support for mathematical operations, complex data structures, and a wider range of support libraries.
- Redevelopment facilitates development of new embedding methods.

- Original design: Tcl provides the user interface layer, with complex data structures and manipulation implemented in C and Fortran.
- **Py-ChemShell: both the user interface and data structures handled at the Python level, while the data structures can be directly accessed by Fortran modules.**

# Py-ChemShell -- Python Implementation

1. User Interface
2. Data Structures
3. Python/Fortran Coupling
4. QM/MM Driver
5. Task-farming Parallelism

# Py-ChemShell -- Python Implementation

## ➢ User Interface

Retains the flexible scripting approach taken by Tcl-ChemShell

Run in Python Interpreter

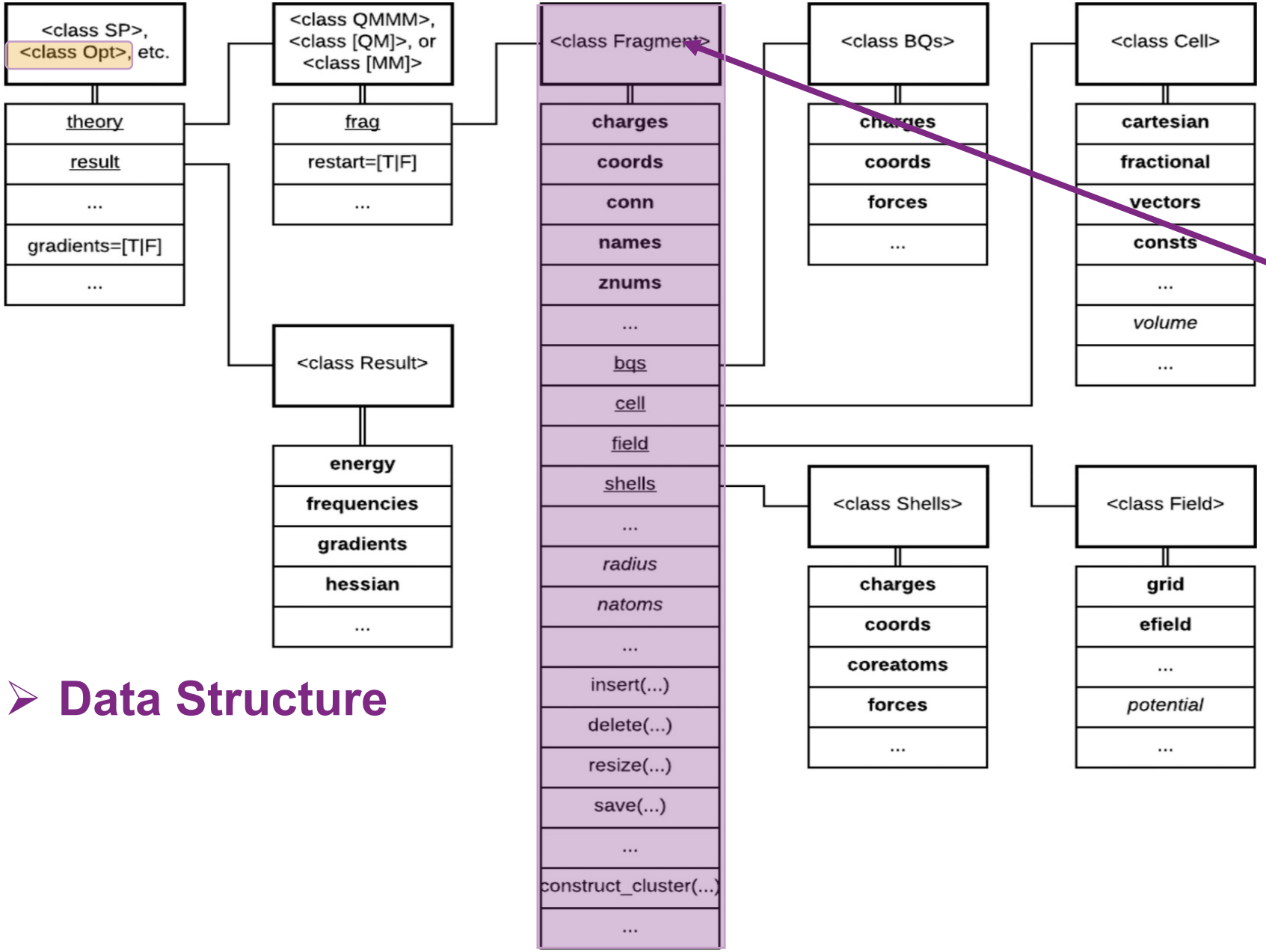
```
# Initialize ChemShell
from chemsh import *
# Load water geometry from XYZ file
my_mol = Fragment(coords='water.xyz')
# Define QM level of theory using NWChem
my_qm = NWChem(frag=my_mol,
method='dft', functional='blyp',
basis='6-31g')
# Run geometry optimization with DL-FIND
my_opt = Opt(theory=my_qm)
my_opt.run()
# Write optimized energy to output
print('Energy = ', my_opt.result.energy)
```

BLYP/6-31G optimization of water using the NWChem QM

Run in ChemShell executable

```
chemsh h2o.py
```

# Py-ChemShell -- Python Implementation




## ➢ Data Structure

```
# Initialize ChemShell
from chemsh import *
# Load water geometry from XYZ file
my_mol = Fragment(coords='water.xyz')
# Define QM level of theory using NWChem
my_qm = NWChem(frag=my_mol,
method='dft', functional='blyp',
basis='6-31g')
# Run geometry optimization with DL-FIND
my_opt = Opt(theory=my_qm)
my_opt.run()
# Write optimized energy to output
print('Energy = ', my_opt.result.energy)
```

Methods to manipulate object data

(~Molecule.delete_atom_by_indices in CHEMSMART)

```
my_mol.delete(0)
```

# Py-ChemShell -- Python Implementation

## ➢ Python / Fortran Coupling

A Fortran layer is required for certain lower level operations.

- Routines for cutting clusters
- The DL-FIND geometry optimization library
- The ability to interface directly to external packages.
- The parallelization framework

**General-purpose library, DL_PY2F**

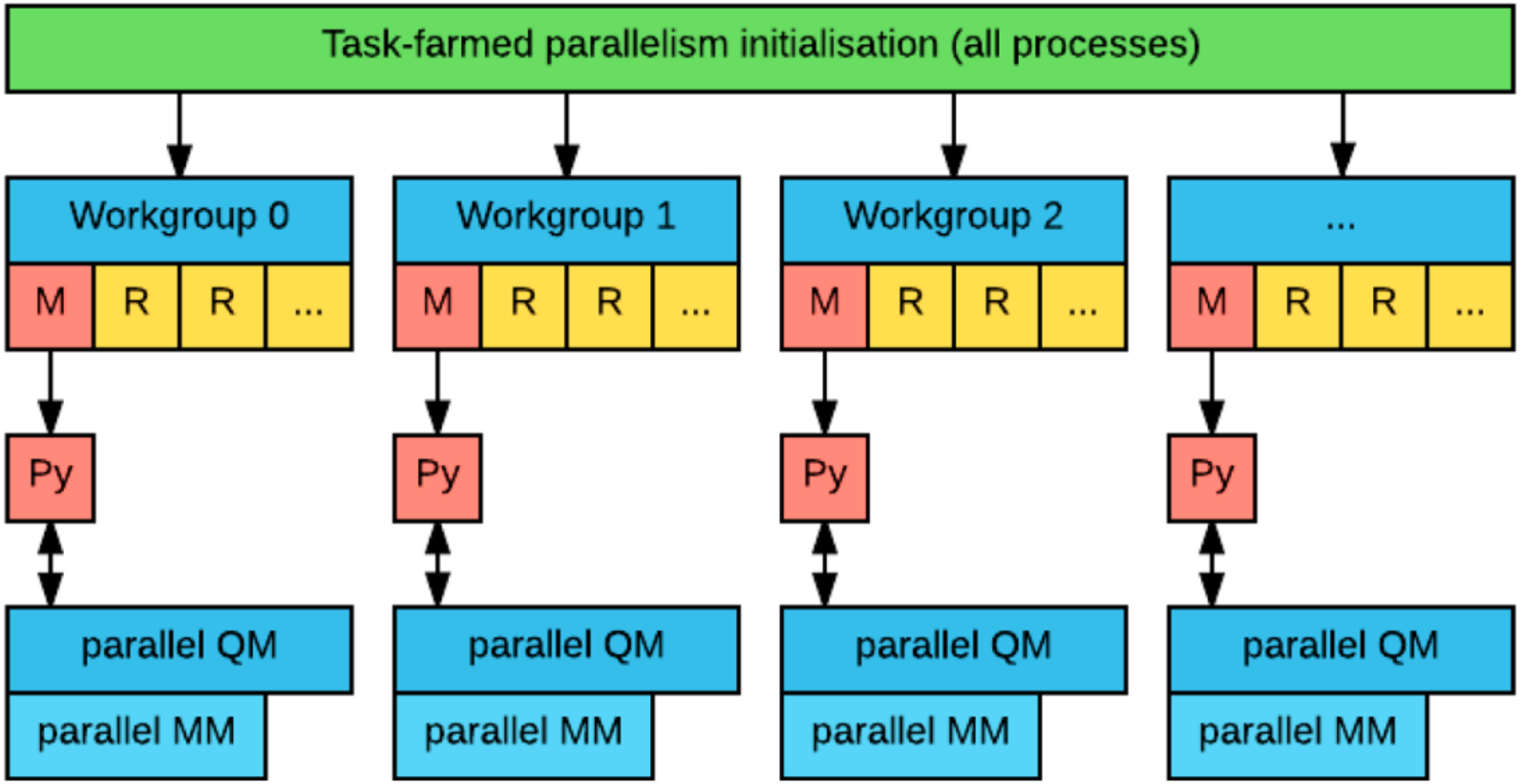
→ enables Py-ChemShell objects to be directly accessed from the Fortran layer

## ➢ QM/MM Driver

- Full support for both link-atom and ionic boundary methods for handling the QM/MM boundary region.
- Support of Mechanical, electrostatic, and polarized (shell model) embedding
- The charge shifting scheme for avoiding overpolarization of electrostatic embedding.
- Link atom forces resolved as in the original implementation.

# Py-ChemShell – Python Implementation

## ➢ Task-Farming Parallelism




- ➢ Independent Parallelism
- ➢ Two-Level Hierarchy
- ➢ QM/MM Resource Efficiency:

  Dynamically allocates processor subsets to minimize communication overhead for smaller MM regions.

Parallel MPI framework in Py-ChemShell following implementation of task-farming parallelization

# Py-ChemShell – Task-Farming Parallelism

## ➢ QM/MM Task-benchmarks

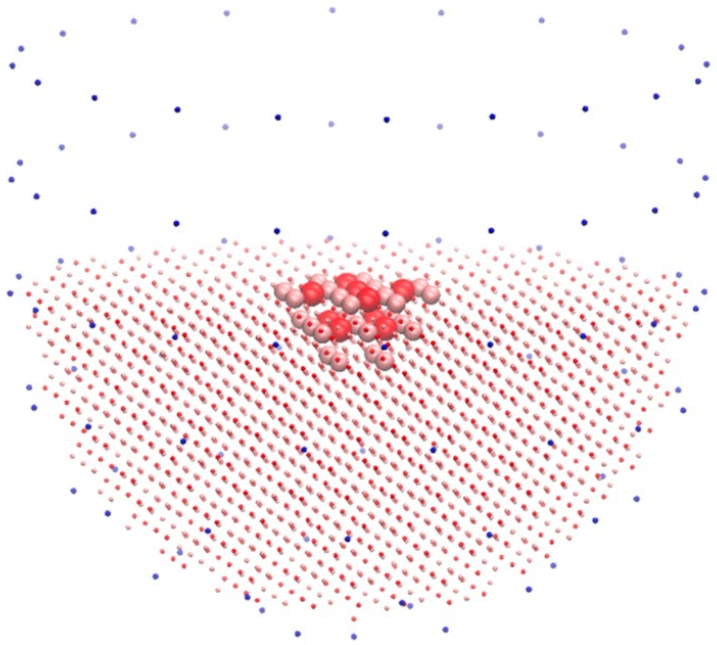


**Figure 3.** QM/MM-embedded MgO system used for two-point finite-difference numerical gradient calculations. The cluster consists of 2263 atoms in total, with the QM region comprising 25 Mg (pink VDW representation) and 9 O (red VDW representation) atoms and the MM region comprising 2229 atoms, with 107 background point charges surrounding the cluster (in blue). Graphic produced with VMD.[92]

**Table 1. Two-Point Finite Difference QM/MM Gradient Task-Farming Benchmark of MgO on a Fixed total of 48 Cores Using GAMESS-UK/GULP**[a]

| $n_{workgroups}$ | 1 | 2 | 4 | 8 | 16 | 24 | 48 |
|---|---|---|---|---|---|---|---|
| $n_{cores}$ | | | | 48 | | | |
| $n_{cores/workgroup}$ | 48 | 24 | 12 | 6 | 3 | 2 | 1 |
| $n_{SP}$ | 412 | 416 | 424 | 440 | 472 | 504 | 600 |
| $t_{SP}$/s (GAMESS-UK) | 2.9 | 4.2 | 5.7 | 9.8 | 18.0 | 25.8 | 57.6 |
| $t_{SP}$/s (GULP) | 2.1 | 2.5 | 2.6 | 3.0 | 4.1 | 5.0 | 8.1 |
| $t_{total}$ (h) | 1.48 | 0.85 | 0.50 | 0.33 | 0.27 | 0.26 | 0.32 |
| speedup vs 1 workgroup | 1.0 | 1.7 | 2.9 | 4.5 | 5.4 | 5.7 | 4.6 |

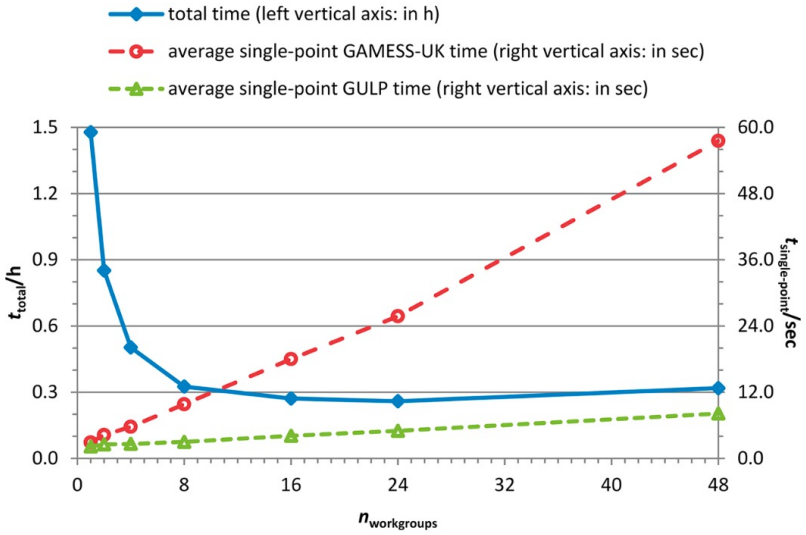



**Table 2. Two-Point Finite Difference QM/MM Gradient Task-Farming Benchmark of MgO for Fixed Size Workgroups Using GAMESS-UK/GULP**[a]

| $n_{workgroups}$ | 1 | 2 | 4 | 8 |
|---|---|---|---|---|
| $n_{cores}$ | 48 | 96 | 192 | 384 |
| $n_{cores/workgroup}$ | 48 | | | |
| $n_{SP}$ | 412 | 416 | 424 | 440 |
| $t_{SP}$/s (GAMESS-UK) | 2.9 | 2.6 | 2.8 | 3.1 |
| $t_{SP}$/s (GULP) | 2.1 | 1.9 | 2.0 | 2.0 |
| $t_{total}$ (h) | 1.48 | 0.72 | 0.37 | 0.20 |
| parallel scaling | 1.00 | 2.05 | 4.04 | 7.26 |

# Py-ChemShell – Task-Farming Parallelism

## ➢ QM/MM nanoparticle benchmarks

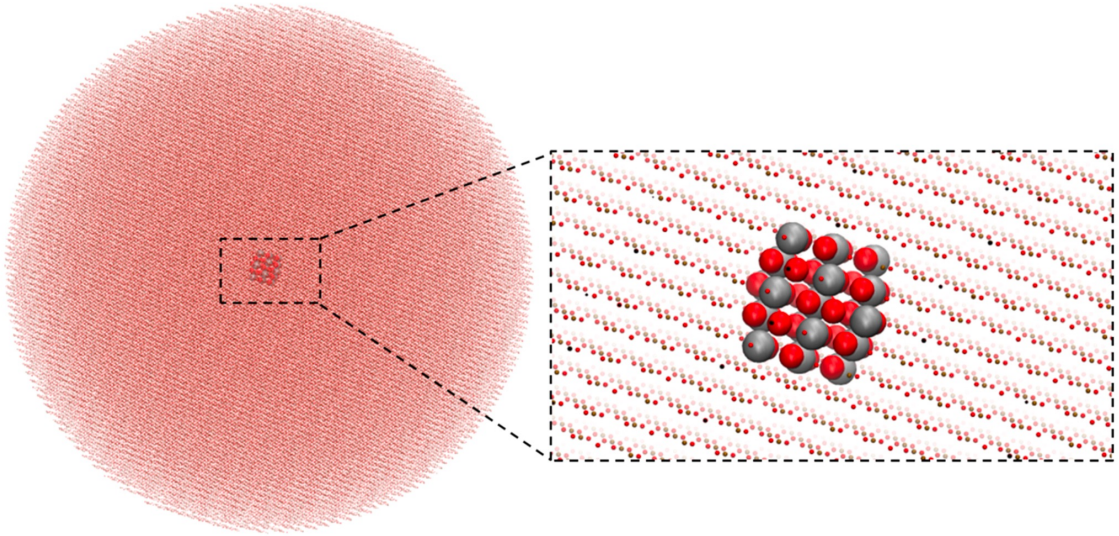


**Figure 6.** $ZrO_2$ nanoparticle of 162994 atoms, including 19 Zr (gray VDW representation) and 32 O (red VDW representation) atoms in the QM region. Graphics generated using VMD.[92]

**Table 3. Single-Point Energy Evaluations of a QM/MM $ZrO_2$ Nanoparticle System of 162994 Atoms Using NWChem/DL_POLY 4**[a]

| $n_{cores}$ | $n_{nodes}$ | $n_{cores}$ (DL_POLY 4) | $t_{NWChem}$ (h) | $t_{DL\_POLY}$ (s) | $t_{total}$ (h) | speedup |
|---|---|---|---|---|---|---|
| 24 | 1 | 8 | 11.99 | 50.1 | 12.21 | 1.00 |
| 48 | 2 | 8 | 6.15 | 49.3 | 6.38 | 1.91 |
| 96 | 4 | 8 | 3.26 | 58.8 | 3.47 | 3.52 |
| 192 | 8 | 8 | 1.71 | 50.8 | 1.92 | 6.36 |

[a]Calculations run on ARCHER.

- System contains 162994 atoms → not possible for Tcl-ChemShell
  (the built-in version of DL_POLY Classic used for MM energy evaluations uses replica data parallelism limited to 50000 atoms)
- Py-ChemShell scales efficiently for this system.

# Py-ChemShell – Recent development

*Version 17.0*
→ targeted on material chemistry

*Version 21.0*[2] supports:

- Biomolecular workflow
- Periodic QM/MM embedding

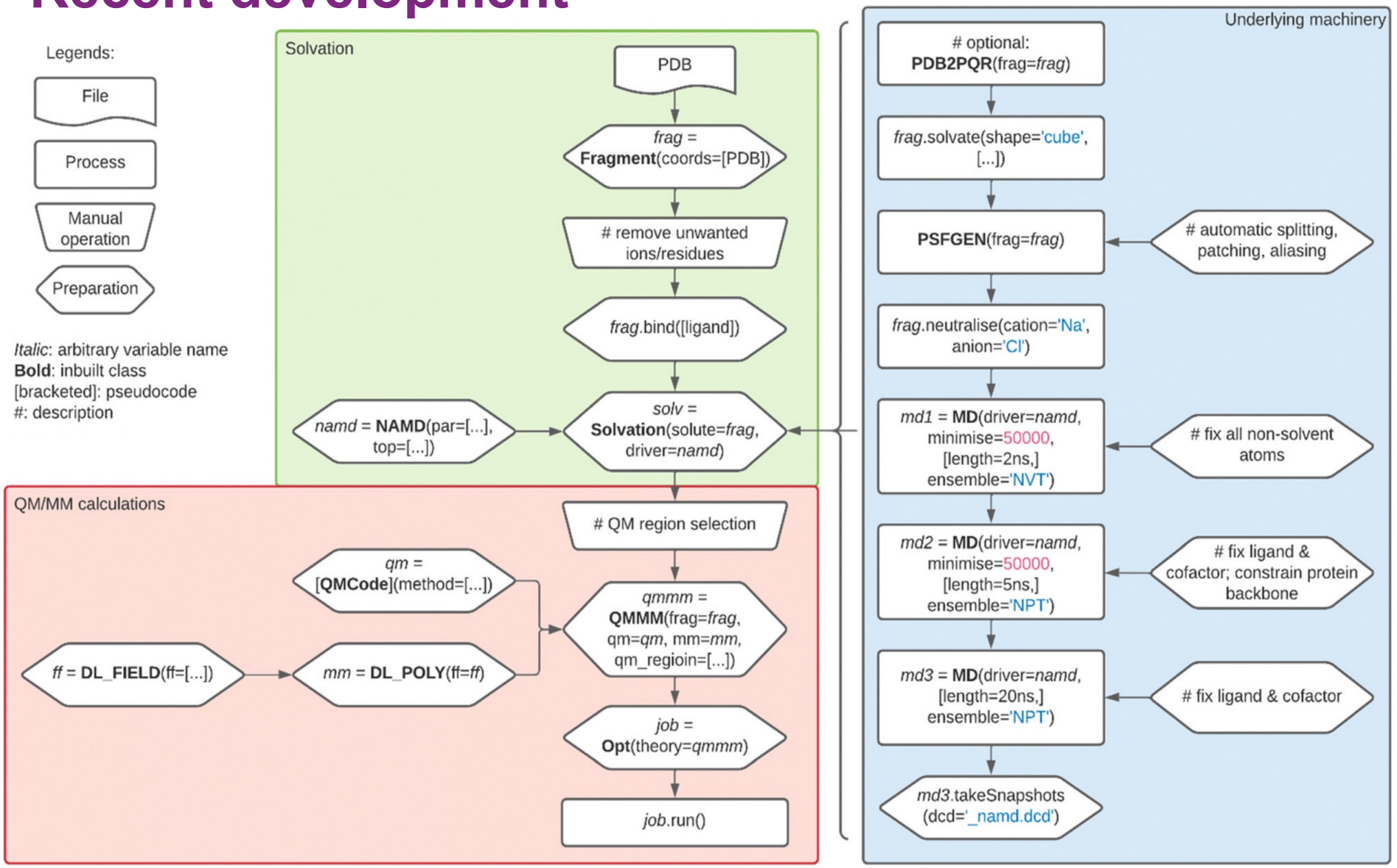



**Fig. 3** Flowchart of the workflow for biomolecular solvation and QM/MM calculations in Py-ChemShell. The underlying machinery shown in the blue box is automated, but may be controlled by the user *via* setting of options at each stage.



[3] Phys. Chem. Chem. Phys. 2023, 25:21816–21835. doi: 10.1039/d3cp00648d

# Py-ChemShell – Recent development

*Version 17.0*
→ targeted on material chemistry

*Version 21.0*[2] supports:

- Biomolecular workflow
- Periodic QM/MM embedding

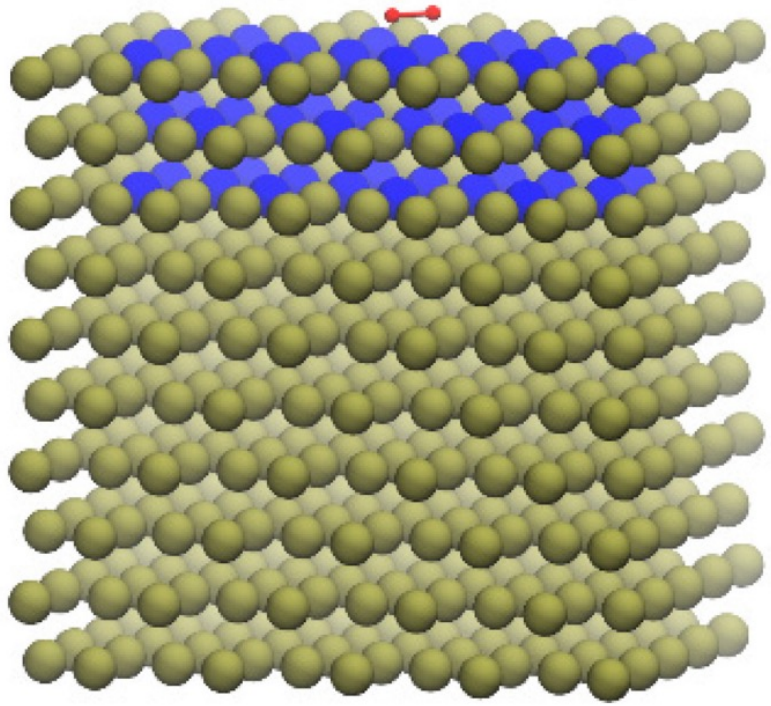


Fig. 6 $O_2$ (red) absorbed on the Pd surface optimised using the QM/Me embedding scheme. The 3-layer QM slab is displayed in blue colour.

- the QM region is calculated with periodic boundary conditions to describe the band structure of the metallic system.
- MM region, is calculated to describe longer range elastic substrate–substrate interactions.
- the QM calculation would also describe substrate–substrate interactions to some extent.
- the QM region is calculated twice: once including both surface and adsorbate and once for the surface alone.

$$E^{\mathrm{QM/Me}}(\boldsymbol{R}) = \left[E^{\mathrm{QM}}(\boldsymbol{R}_{\mathrm{slab}} \cup \boldsymbol{R}_{\mathrm{ads}}) - E^{\mathrm{QM}}(\boldsymbol{R}_{\mathrm{slab}})\right] + E^{\mathrm{Me}}(\boldsymbol{R}_{\mathrm{slab}} \cup \boldsymbol{R}_{\mathrm{env}})$$



[3] Phys. Chem. Chem. Phys. 2023, 25:21816–21835. doi: 10.1039/d3cp00648d

# Conclusion

## ➢ Current Capabilities

•Py-ChemShell combines the modern Python interface with a rugged, highly parallelized Fortran computational backend.

•Fully supports classical biomolecular workflows alongside materials chemistry workflows.

•Features integrated boundary cluster models and interfaces with packages like ORCA, DFTB+, CP2K, and CRYSTAL.

## ➢ Future development

- Launching adaptive solvent-exchange QM/MM, PLUMED free energy metadynamics, and IR/Raman spectroscopy profiles.
- Merging ML potentials with geometry optimization drivers.
- Rewritten under ExCALIBUR to tackle highly scalable multiscale simulation workloads on tomorrow's supercomputers.

# Questions? Comments?

# Thank You